CS 6240 – Section 2
HW 5
Darshan Patel
Patel.dars@husky.neu.edu

Design Discussion and Pseudo-Code

Matrix Representation:

I have used matrix representation in which each column represents a page and column values represent its page rank contribution to other pages/rows (Here rows also represent pages).

For example:
A -> [B, C]
C -> [A]

The above page name and its adjacency pages would be represented as:

$$M = \begin{bmatrix} & A & B & C \\ A & 0 & 0 & 1 \\ B & \frac{1}{2} & 0 & 0 \\ C & 1 & 0 & 0 \\ & \frac{1}{2} & & \end{bmatrix}$$

How matrix is saved for Version A?

In version A, we have row major matrix multiplication, so I have saved matrix as sparse matrix it terms of tuple. The format of tuple is (row id, (column id, page rank contribution to row id page))

For the above example, it would be let's say ids are like (A <- 1, B <- 2, C <- 3)
(1, (3, 1))
(2, (1, 1/2))
(3, (1, 1/2))

How matrix is saved for Version B?

In version B, we have column major matrix multiplication, so I have saved matrix as sparse matrix it terms of tuple. The format of tuple is (column id, (row id, page rank contribution from column id page))

For the above example, it would be let's say ids are like (A <- 1, B <- 2, C <- 3)
(1, (2, 1/2))
(1, (3, 1/2))

(3, (1, 1))

Step:1 Page Names to Ids

To assign id to each page name, I have used two map reduce jobs. The first map reduce job parse the input data and converts into adjacency list and second map reduce job assign the unique id to each page. Below pseudo code is same for both the version of the program.

The first job parses the input and makes the adjacency list out of it. Apart from that it also converts the dead node into dangling node. I have used combiner to reduce the data flow between the machines. To convert dead node into dangling, I have emitted each page in adjacency list as key and empty/null adjacency list as value.

Job 1: Parse Data and convert dead nodes into dangling nodes

map (…, record r)

```
//Parse given html record r and extract out link page names
Node.adjNodes <- out link pages
emit (page name, Node)
//To convert dead nodes in adjacency list into dangling nodes
foreach adjacency page in Node. adjNodes:
        emit (adjacency page, None)
```

partitioner (page name, Node, number of reducer)

```
return max(pagename.hashCode() % number of reducer, 0)
```

combiner (page name, [node1, node2, node3, …])

```
// Get only first not null node from [node1, node2, node3, …] if exists otherwise emit Null
emit (page name, node1)
```

reduce (page name, [node1, node2, node3, …])

```
// Get only first not null node from [node1, node2, node3, …] if exists otherwise emit node
with empty adjacency list.
emit (page name, node1)
```

Job 2: Assign unique id to each page.

I have assigned the unique id to each page using the task id and total number of task for the given job.
Example: Let's say there are four task with ids: 1, 2, 3, 4
For the pages in task 1 the unique id would be: 1+4, 1+4+4, 1+4+4+4 -> 5, 9, 13
For the pages in task 2 the unique id would be: 2+4, 2+4+4, 2+4+4+4 -> 6, 10, 14

This way each page will receive unique ids.

Mapper:

setup ()

taskID <- get current task id
increment <- total number of task for the given job by Map Reduce framework.
// Open multiple outputs

map (…, record r)

unique id <- taskId + increment
// Multiple outputs
if (r has empty adjacency list)
    emit ('danglingIDs', page name, unique id)
emit ('IDs', page name, unique id)

cleanup ()

close multiple output

No reducer

Step:2 Build Matrix from adjacency list:

Once the unique ids are assigned to each page, the result is stored in the output files. For the next map reduce job, I have used those files as cache file and loaded it as HashMap in the setup method of each mapper.

In each map call, I have used that HashMap to convert page name into page id and emitted in the following format (row id, (column id, page rank contribution))

Version A:

setup ()

// Load Map Reduce Job 2 output as cache file
// Read the cache file and converts into Hash Map where key is page name and value is
// unique id.

map (page name, Adjacency pages)

length <- number of pages in adjacency list
for each adjacency page in Adjacency pages:
    // (row, (column, contribution)) as per the matrix representation I have used.
    Cell.index <- HashMap.get(adjacency page)
    Cell.key <- 0
    emit (Cell, (HashMap.get(page name), 1/length))

No reducer

Version B:

setup ()

```
// Load Map Reduce Job 2 output as cache file
// Read the cache file and converts into Hash Map where key is page name and value is
// unique id.
```

map (page name, Adjacency pages)

```
length <- number of pages in adjacency list
for each adjacency page in Adjacency pages:
    // (column, (row, contribution)) as per the matrix representation I have used.
    Cell.index <- HashMap.get(page name)
    Cell.key <- 0
    emit (Cell, (HashMap.get(adjacency page), 1/length))
```

No reducer

The difference between the above two versions is highlighted above. In the version A, row id (The page to whom the current page contributes) is emitted as key and in the version B, the source page who contributes to the other page is emitted as key.

Step:3 Page Rank Calculation

Version A:

To perform the $M'R$, I have converted M' into $M + D$. Matrix D contains value (1/N) at those columns which represent dangling page. Here (1/N) is page rank contribution of the dangling page to others pages.

To simplify the process to compute D*R, I have already created list of dangling node in Job 2, which I have read it as cache file along the file which contains the initial page rank of each node. (For the iteration 2 onwards it would the output file of previous iteration)

In setup method of each reduce, I sum the page rank of each dangling node and multiple the sum with (1/N). Which is equivalent of multiply row of D with column of R. Now this constant would be added in each M*R resulted row.

Here I have used identity mapper so the reduce will receive the data related to each row. Now to calculate M*R. R is already loaded from the cache, I have multiplied the values of each received column for the given row (key) with the corresponding row of the page rank matrix and summed them up to calculate the new page rank.

Once the M*R is calculated, I add D*R which is already calculated in the setup method of each reducer.

Note: The only issue was the pages which do not have any in links so there won't be a reduce call for those pages and they won't appear in the output files. To handle this issue, I have created a mapper which emits each page id with its page rank and takes the page rank matrix as input. The other mapper is identity mapper.

To distinguish rank mapper entry, I have applied secondary sort. In the matrix building job I have given cell key as 0 but now for the rank mapper I will give cell key as -1 and you that attribute for the secondary sort.

Mapper:

map (record r)

```
cell.key <- (-1)
cell.index <- r(page id)
emit (cell, page rank)
```

Grouping Comparator

```
Compare (cell c1, cell c2)
    Return c1.index.compareTo(c2.index)
```

Cell

```
compareTo(Cell c)
    // Compare index first if they are not same
    // then compare key
    if index same:
        return key comparison
    else:
        return index comparison
```

Reducer:

setup ()

```
    // Build PageRank HashMap
    if (iteration == 1)
        //Load Map Reduce Job 2 output (Page Id assignment) as cache file and converts
        // into hash Map where key is page id and value is initial page rank.
    else
        //Load Previous Map Reduce Job output as cache file and converts into hash Map
        // where key is page id and value is page rank.
```

```
// Calculate sum of dangling node page rank.
// Load Map Reduce Job 2 output (Dangling nodes) as cache file
danglingPageRanksum = 0
for each dangling node:
        sum += pageRank.get(dangling node)

// D * R
danglingPageRanksum  = (1/N) * danglingPageRanksum
```

reduce (row id, [(column id, page rank contribution), …])

```
//First record would be from rank mapper due to secondary sort so just ignore

pageRankContributionSum = 0
for each column id, page rank contribution [(column id, page rank contribution)]
        pageRankContributionSum  += pageRank.get(column id) * page rank contribution

// Add the D*R by adding  danglingPageRanksum
pageRankContributionSum += danglingPageRanksum

// Calculate new page rank
new page rank <- (alpha/N) + (1-alpha) * pageRankContributionSum
emit (row id, new page rank)
```

Version B:

I have used two Map Reduce Jobs to calculate the $M'R$

First Job:

The first job has two mappers. One of the mapper is identity mapper which reads the output of step-2 for the version B which is (column id, (row id, contribution))

The second mapper takes the page rank matrix as input and emits (row id, page rank).

The reducer receives the data associated with each column. The data is received by reducer in sorted using secondary sort so the page rank will be received first (which is second map output).

Now the received page rank would be distributed to the all the row id for the current column id. So, the reducer will emit (row id, page rank contribution).

Mapper:

map (record r)

cell.key <- (-1)
cell.index <- r(page id)
emit (cell, page rank)

Grouping Comparator

Compare (cell c1, cell c2)

Return c1.index.compareTo(c2.index)

Cell

compareTo (Cell c)

// Compare index first if they are not same
// then compare key
if index same:
    return key comparison
else:
    return index comparison

Reducer

setup ()

Set up HashMap for in-reducer combining.

reduce (column id, [(row id, page rank contribution), …])

//First record would be from rank mapper due to secondary sort
if first record:
    //extract page rank
else:
    for each row id in **[(row id, page rank contribution), …]**
        // Sum the page rank contribution for the given row id
        HashMap (row id) += (page rank contribution * page rank)

If column id doesn't exist
    HashMap (column id) = 0

Cleanup ()

For each entry in hash map:
    Emit (index, page rank contribution sum)

Second Job:

Second job receives all the page rank contribution for the given row of the page rank matrix. Now just sum the page rank contribution and calculate the new page rank and emit row id and page rank of that row id for the page rank matrix.

Mapper

Setup ()

```
// Set up Hash Map where key is row id and value is page rank contribution
```

map (record r)

```
HashMap (r.row id) += r.page rank contribution
```

Cleanup ()

```
For each entry in hash map
        Emit (row id, page rank contribution)
```

Combiner

map (row id, [page rank contribution 1, page rank contribution 2, …])

```
total page rank contribution = 0

for each page rank contribution in [page rank contribution 1, …])
        sum them
emit (row id, sum)
```

Reducer

setup ()

```
// Build PageRank HashMap
if (iteration == 1)
        //Load Map Reduce Job 2 output (Page Id assignment) as cache file and converts
        // into hash Map where key is page id and value is initial page rank.
else
        //Load Previous Map Reduce Job output as cache file and converts into hash Map
        // where key is page id and value is page rank.

// Calculate sum of dangling node page rank.
// Load Map Reduce Job 2 output (Dangling nodes) as cache file
danglingPageRanksum = 0
for each dangling node:
        sum += pageRank.get(dangling node)

// D * R
danglingPageRanksum = (1/N) * danglingPageRanksum
```

reduce (row id, [page rank contribution, …])

// Sum page rank contributions
// Add dangling node page rank contribution into total page rank contributions
// Calculate new page rank

emit (row id, new page rank)

Step:4 Top - K

Mapper:

Setup ():

Max heap with capacity 100

Map (record r):

Add into max heap

Cleanup ():

For each element in max heap:
    Emit (page rank, page id)

Reducer:

Setup ()

// Build PageRank HashMap

//Load Map Reduce Job 2 output (Page Id assignment) as cache file and converts // into hash Map where key is page id and value is initial page name.
counter <- 1

reducer (page rank, [(page id, page ids, …)])

until counter < 100:
    page name <- hashmap.get(page id)
    emit (page name, page rank)
    counter += 1

Dangling Node Handling:

Version A:

I have divided the matrix M'into two parts (M and D). D only contains dangling node contribution to other pages which is (1/N). I have used distributed cache file to calculate D *R. As

each row in D is equal the multiplication of D and R would have n by 1 matrix with same value. Which would be added into the result of M *R.

In each reducer setup method, I have read dangling node ids and summed their page rank and divided page rank sum by N. After that in the reduce call, to calculate the new page, that calculated D * R number would be added into the sum of page rank contribution and new page rank would be calculated.

This solution doesn't need additional map reduce job.

Version B:

Its handled same way as version A but the different is in D * R calculation as D is considered as column major matrix. Apart from that I haven't used any additional map reduce job.

Performance Comparison

Step Number	Row Major		Column Major	
	6 Machines	**11 Machines**	**6 Machines**	**11 Machines**
1 and 2	28 Minutes	17 Minutes	31 minutes	17 Minutes
3 (Per iteration)	9 Minutes	6 Minutes	10 minutes	12 Minutes
4	1:43 Minutes	1:40 Minutes	1:47 minutes	1:55 Minutes

Adjacency based approach Results

	6 Machines	11 Machines
Step 1/2	42:48 minutes	17:75 Minutes
Step 3	19:10 Minutes overall	12:59 Minutes overall
Step 4	00:37 Minutes	00:45 Minutes

Conclusion:

The difference is significant as each iteration in the column major matrix based approach takes 10-12 minutes compared to 18 minutes in all iteration for adjacency list.

The reason why matrix based perform is worst is due to my implementation mistake. Apart from that, I believe either row or column matrix based approach should outperform the adjacency based approach.

As only single job is needed in row based approach which should be fast enough to calculate compare to two map reduce job for the column based approach.

Top 100

Full Dataset

United_States_09d4 0.0026227969707326576
2006 0.0012284657121117717
United_Kingdom_5ad7 0.0012030960092862335
Biography 9.82048251703487E-4
2005 9.170121506132236E-4
England 8.801716585086553E-4
Canada 8.558814015008397E-4
Geographic_coordinate_system 7.716405680234864E-4
France 7.249865560476987E-4
2004 7.198567611786041E-4
Australia 6.804534373356086E-4
Germany 6.543273669808579E-4
2003 5.873770965110631E-4
India 5.833996611232281E-4
Japan 5.828276092629625E-4
Internet_Movie_Database_7ea7 5.334890996115478E-4
Europe 5.092533549845222E-4
Record_label 4.914259668422064E-4
2001 4.8700487428671044E-4
2002 4.828613288590205E-4
World_War_II_d045 4.780321448107464E-4
Population_density 4.703281517593577E-4
Music_genre 4.671790567006932E-4
2000 4.6465230412643483E-4
Italy 4.457816775151184E-4
Wiktionary 4.361459118897651E-4
Wikimedia_Commons_7b57 4.352824154525449E-4
London 4.34779811184388E-4
English_language 4.1847571750759416E-4

1999 4.0592257521267363E-4
Spain 3.629179658237518E-4
1998 3.562950263938605E-4
Russia 3.438825297702386E-4
1997 3.372705528751522E-4
Television 3.362857525898893E-4
New_York_City_1428 3.34617291374476E-4
Football_(soccer) 3.261384037874272E-4
1996 3.236137343078874E-4
Census 3.235439859381969E-4
Scotland 3.2218004198659E-4
1995 3.1014423071102005E-4
China 3.0862741664180706E-4
Population 3.0430250807770925E-4
Square_mile 3.0404541163134296E-4
Scientific_classification 3.0400099824082693E-4
California 3.0165682158566135E-4
1994 2.9067855520458277E-4
Sweden 2.8761079378259835E-4
Public_domain 2.874097255101378E-4
Film 2.862605348883609E-4
Record_producer 2.840935434084946E-4
New_Zealand_2311 2.8309219236856274E-4
New_York_3da4 2.788771244951798E-4
Netherlands 2.766591971087185E-4
Marriage 2.7580395938349645E-4
1993 2.747885229183738E-4
United_States_Census_Bureau_2c852.7465767508117627E-4
1991 2.7189466968382604E-4
1990 2.6831508084560087E-4
1992 2.66353294846011E-4
Politician 2.6488541446855905E-4
Album 2.6055477814534593E-4
Latin 2.604473331218022E-4
Actor 2.5833050910300913E-4
Ireland 2.581015993822455E-4
Per_capita_income 2.5563401652518733E-4
Studio_album 2.518496911745388E-4
Poverty_line 2.511564885522393E-4
Km² 2.494985147573982E-4
1989 2.468778672331587E-4
Norway 2.4096337428979283E-4
Website 2.3900362628144858E-4
1980 2.353149412908127E-4

Animal 2.2937018830850581E-4
Area 2.2919782132795952E-4
1986 2.2702603702726034E-4
Personal_name 2.2623624857475878E-4
Poland 2.2610997030180827E-4
Brazil 2.2571993204975938E-4
1985 2.240212545454565E-4
1987 2.2329776410713755E-4
1983 2.2174577230568108E-4
1982 2.210905978799829E-4
French_language 2.193741556513367E-4
1981 2.1933979554486366E-4
1979 2.193231103594094E-4
1984 2.1878073332143555E-4
World_War_I_9429 2.1868452447728885E-4
1988 2.185670859964644E-4
Paris 2.1801194233307705E-4
1974 2.1796821432217882E-4
Mexico 2.1566084610419718E-4
19th_century 2.118499255333935E-4
1970 2.1131665914116905E-4
January_1 2.1085145665723798E-4
USA_f75d 2.1070160934467797E-4
1975 2.085941796284711E-4
1976 2.0846090428633642E-4
Africa 2.07792917822948E-4
South_Africa_1287 2.0735155711845482E-4

Local Dataset

United_States_09d4 0.005204434652932262
Wikimedia_Commons_7b57 0.004848858587766546
Country 0.003981569654898079
England 0.0027601270839197954
Water 0.0026939426537260667
City 0.00255830359059524 7
Animal 0.002555785303166791
Germany 0.0024187103351786306
United_Kingdom_5ad7 0.0023668080812607136
France 0.0023481574074686764
Earth 0.0023318206811890987
Europe 0.0020539749518054488
Wiktionary 0.0017571609290764935
English_language 0.0017561267715258278

Government 0.0017445497720584987
India 0.0017204323804524628
Computer 0.0017160467379080384
Money 0.0016713593017605817
Japan 0.0015579163067532085
Plant 0.0015234451665910285
Italy 0.0015157302947336807
Canada 0.0014903846946996552
Spain 0.0014898737384772451
Food 0.0014239241517878005
Human 0.0014163100855977065
China 0.0014019523971592499
People 0.00138706084455148
Australia 0.0013329561540502217
Asia 0.0012910719500721735
Capital_(city) 0.0012881605799375951
Television 0.0012662962859795795
Sun 0.00126146165522533
State 0.001251223699234101
Number 0.001243669907057066
Sound 0.0012405237622490075
Science 0.0012338430146784807
Mathematics 0.00123165413048998
Metal 0.0011927212867573
2004 0.0011854690140499324
Year 0.0011795526503280709
Language 0.00115836476043548
Russia 0.0011513193861339014
Wikipedia 0.0011295158232662939
19th_century 0.0011027255877008476
Religion 0.0011017818146793706
Music 0.0010888431242898342
20th_century 0.0010605465591555178
Scotland 0.0010575869456937953
Greece 0.0010533989382284033
Latin 0.0010351507611641127
London 0.0010287176944132545
Greek_language 0.001008918662164986
Energy 9.993856141285221E-4
World 9.912736577364393E-4
Centuries 9.78618549477999E-4
Culture 9.513850045853615E-4
History 9.441133531805147E-4
Liquid 9.14518136193652E-4

Netherlands 9.106068003548995E-4
Planet 9.075892884371957E-4
Society 9.043801282980952E-4
Light 9.019502000417065E-4
Wikimedia_Foundation_83d9 8.94205975182407E-4
Image 8.935670681288561E-4
Atom 8.890928075608078E-4
Law 8.886976175265509E-4
Scientist 8.886551254216366E-4
Geography 8.853120249178912E-4
List_of_decades 8.814789976308119E-4
Africa 8.757049806880343E-4
Uniform_Resource_Locator_1b4e 8.675976070867932E-4
Inhabitant 8.559517974315178E-4
Turkey 8.507120596010292E-4
Capital_city 8.373158271108687E-4
Poland 8.314098569104419E-4
Plural 8.239476187924812E-4
Electricity 8.141098963453186E-4
Building 7.98853558399124E-4
Car 7.970192192817732E-4
Sweden 7.96780439428771E-4
Book 7.922095258977724E-4
Biology 7.861009718530094E-4
War 7.746729095311604E-4
Chemical_element 7.669135649386006E-4
God 7.66102916408532E-4
September_7 7.602904268996228E-4
North_America_e7c4 7.589071492620048E-4
Nation 7.488746260982865E-4
Website 7.487914549859853E-4
Politics 7.447765868320805E-4
2006 7.38263682883684E-4
Switzerland 7.350515016004583E-4
Fish 7.331154911057659E-4
Species 7.29620611678403E-4
Portugal 7.244801260968459E-4
River 7.220523282047657E-4
Mammal 7.215886644055932E-4
Island 7.202533904727492E-4
Gas 7.15763198939511E-4
World_War_II_d045 7.086034184791898E-4

Conclusion:

This results are almost identical to the result I have received for the adjacency based page rank calculation. The difference may be due to my mistake in implementing adjacency based method or map reduce calculation precision.